# The use of computer-based programming environments as computer modelling tools in early science education: the cases of textual and graphical program languages

Louca, Loucas; Zacharia, Zacharias C.



**Empfohlene Zitierung / Suggested Citation:**
Louca, L., & Zacharia, Z. C. (2008). The use of computer-based programming environments as computer modelling tools in early science education: the cases of textual and graphical program languages. *International Journal of Science Education*, *30*(3), 287-323. https://doi.org/10.1080/09500690601188620


# The Use of Computer-Based Programming Environments as Computer Modelling Tools in Early Science Education: The Cases of Textual and Graphical Program Languages

| | |
|---|---|
| Journal: | *International Journal of Science Education* |
| Manuscript ID: | TSED-2006-0229 |
| Manuscript Type: | Research Paper |
| Keywords: | science education, model-based learning, elementary school |
| Keywords (user): | Program Languages |
| | |

# The use of computer-based programming environments as computer modelling tools in early science education: the cases of textual and graphical program languages


## Abstract

This is an interpretive case study seeking to develop detailed and comparative descriptions of how two groups of fifth grade students used two different Computer-Based Programming Environments (CPEs) (namely Microworlds Logo and Stagecast Creator) during scientific modelling. The primary sources of data that were used in this four-month-long study include videotaped students' group work and whole-class discussions, and the instructors' reflective journals. For the data analysis contextual inquiry was used in conjunction with analysis of student conversation in order to gain better insight in students' activity and conversation patterns while working with CPEs. Findings highlight the differences in the ways that the students used the two CPEs in the context of developing models of natural phenomena with respect to three distinct phases that emerged from data analysis that include student approaches to (i) planning, (ii) writing and debugging code and (iii) using code to represent the phenomenon under study. Lastly, findings highlight which aspects of students work during the three phases can be productive for scientific modelling, proposing possible relationships between student work and CPE features.

# The use of computer-based programming environments as computer modelling tools in early science education: the cases of textual and graphical program languages

## Introduction

Models are human constructs of systematic representations of a system, or of some simplified part(s) of a system, that include rules and relations between objects, physical values and physical concepts (Glynn & Duit, 1995; Gilbert, 1995; Ingham & Gilbert, 1991) seeking to provide a representation for the mechanism that underlies the natural phenomena in a coherent way. They are used to describe, represent and explain the mechanisms underlying natural phenomena, having both explanatory power (Gilbert et al, 1998) and predictability for those phenomena (Erduran, 1999; Gilbert, 1995; Gobert & Buckley, 2000). In addition, models are also used to make abstract entities visible (Francoeur, 1997) and provide a basis for interpreting experimental data and results (Tomasi, 1988). Good models extend across individual systems and are complete descriptions of our understanding of fundamental mechanisms in nature. In this sense, a natural system can be modelled by identifying the objects of the system, the functions or behaviours of each object and the relationships among these objects or their behaviours (Constantinou, 1996).

Researchers have presented models and the process of scientific modelling as core components of science education (diSessa, Abelson, & Ploger, 1991; Justi & Gilbert, 2002; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Frederiksen, 1998; Wilensky & Resnick, 1999), not only because the heart of learning in science is the construction and use of models of natural phenomena, but also because part of the learning in science entails learning with and

about the process of scientific modelling (Linn, 2003). Science proceeds through the construction and refinement of models (Constantinou, 1996), and therefore learning science should include developing understanding about natural phenomena by constructing models, as well as learning the process of developing and refining those models (National Research Council, 1990; White & Frederiksen, 1998). In this view, models can play a dual role in science learning: they can be both tools for learning and learning outcomes.

## Modelling-based learning in science

Modelling-based learning can provide the context in which the development and refinement of models can achieve better quality outcomes in terms of fundamental understanding of concepts, operational understanding of the nature of science and the ability to employ procedural and reasoning skills, than what is currently possible through other learning environment/tool in many educational systems (Harrison & Treagust, 1998; Bell, 1995; Grosslight, Unger, Jay & Smith, 1991). Moreover, any learning experience that is grounded upon the premises of modelling-based learning offers students, through an authentic inquiry-oriented practice, an opportunity to think and talk scientifically about natural phenomena (Penner, 2001), to share, discuss and criticize their ideas (Devi, Tiberghien, Baker, & Brna, 1996; Rouwette, Vennix, & Thijssen, 2000; Suthers, 1999) and to reflect upon their own understanding (Gilbert, Boulter, & Rutherford, 1998; Jonassen, Strobel, & Gottdenker, 2005). Penner (2001) has argued that models can be "tools to think with and to reflect upon", because they include representations of physical and conceptual values that are not usually represented in "concrete" forms and therefore cannot be otherwise observed in the natural world (p. 2).

Research in science education has highlighted a number of modelling-based learning approaches (that is the construction of models through the process of scientific modelling) in science (see Justi & Gilbert, 2002, for a review; also see Constantinou, 1996; Penner, 2001; Penner, Lehrer, & Schauble, 1998; Schecker, 1993; Gobert & Buckley, 2000; Glynn et al, 1994; Treagust et al, 1996; Pittman, 1999; Iddling, 1997; Gilbert, 2004). Nevertheless, all researchers appear to identify that the modelling-based learning approach involves two basic steps. The first step is the identification of the need to describe, predict and/or explain a natural phenomenon, which will then guide the learners to investigate the phenomenon and develop a model to represent it. The learners use their experiences (such as observations from everyday life or laboratory-based experiences) to simplify the natural world into objects and their interactions to be represented in a model (Constantinou, 1996; Schecker, 1993).

The second step of the modelling-based learning approach is the evaluation of the model. Once students have constructed a model, they need to evaluate their model through a comparison with the real-life phenomenon (Bell, 1995; Papaevripidou, Constantinou and Zacharia, in press; Penner, 2001; Penner, Lehrer & Schauble; 1998; Schecker, 1993; Gobert & Buckley, 2000). Students should attempt to apply their model to new situations by using the model to interpret and make predictions about new phenomena. This evaluation of their model would lead to subsequent modification(s) of the model, if needed. Studies focusing on modelling-based learning have shown to engage students in the authentic practice of using models as tools for observation, exploration, synthesis and prediction and to provide a learning environment where learners can be engaged in the processes of science through

building, testing, revising and applying models (Papaevripidou, Constsantinou and Zacharia, in press; Schwartz and White 2005).

Modelling-based learning is highly related to the modelling tool used (drawings, mathematical equations, graphs, three-dimensional structures, computer-based programming media and computer-based modelling environments or even words). Hence, one important factor/parameter that should be considered before implementing modelling-based learning within a learning environment is the modelling tool itself. The quality and functionality of a model depends upon the representation medium that is used to represent and develop the model of a natural phenomenon. Consequently, the degree of how well students conceptualize natural phenomena varies according to the modelling tool used to construct and communicate a model to others (Papaevripidou, Constantinou and Zacharia, in press). The most promising educational modelling tools that appear in the literature are computer-based.(Kurtz dos Santos and Ogborn 1994; diSessa, Abelson, Ploger, 1991; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Fredriksen, 1998; Wilensky & Resnick, 1999: Louca, 2004).

*Computer-based Modelling Tools*

A computer-based modelling tool consists of an open-ended, dynamic and exploratory learning environment which among others supports the construction of representation of complex phenomena or natural systems through the simultaneous application/execution of multiple processes in order to go beyond static representations or static structural depictions to dynamic representations of cause/effect relationships among variables (Sins, Savelsbergh and van Joolingen,

2005). In addition, it allows students to visualize abstract concepts (Barab, Hay, Barnett and Keating, 2000) and complex relationships (Singer, Krajcik and Marx, 2000). This latter feature is very important for learning in science because it can enable learners to overcome some of the conceptual and reasoning difficulties they face when studying complex systems.

Currently, a large number of computer-based modelling tools are available and suitable for educational purposes. Despite their similarities, most of these tools have unique characteristics that differentiate them from others, thus, making their selection for a particular modelling assignment a challenging task. Research thus far has failed to establish the criteria (e.g., interface, modelling language, availability of tools/features) that should be used for the selection of the most suitable computer-based modelling tool given a specific age-group and/or a particular natural phenomenon/system.

The current study aimed to contribute towards this direction by investigating how two groups of fifth-grade elementary school students used two different computer-based modelling tools to develop models of natural phenomena. Specifically, this study aimed to investigate the use of a particular family of computer-based modelling tools, namely, computer-based programming environments (CPEs) that research has confirmed their importance of being used as tools for teaching practices of modelling and science (Louca, submitted; Louca, 2004; Louca, 2005). CPEs provide a microworld environment that has no rules and follows no physical laws, and provide a program language as the modelling tool for developing representations of natural phenomena. In contrast to other computer-based modelling

tools that can be used only for the construction of symbolic simulations (models), CPEs enable users to develop "concrete" simulations of natural phenomena/systems that can include animation-like representations of those phenomena/systems that are result of the program code. Our decision for using CPEs as modelling tools lies on the idea that the process of scientific modelling can be compared to the process of computer programming, and modelling can be carried out through developing a computer program, when the program itself becomes the scientific model.

Programs in CPEs produce *a computer microworld* which is a structured environment that learners can use to explore and manipulate a rule-generated universe, subject to particular assumptions and constraints that serve as representations of aspects of the natural world (Pea, 1984). Computer microworlds are idealized environments composed of a collection of objects, relationships among objects and operations that transform the objects and their relationships, all of which are represented in well-specified rules (Thompson, 1985; Miller, Ogborn, Briggs, Brough, Bliss, Boohan, Brosnan, Mellar and Sakonidis, 1993). Microworlds can also provide learners with opportunities to manipulate realities in ways that learners cannot do with physical objects (diSessa, 1982; 1988). Thus, when a computer program, for example, becomes a "thing" that is named, it can be readily manipulated and recognized by students (Papert, 1980).

Currently there are a number of widely-varying CPEs designed for young learners including textual programming [Microworlds Logo, Papert, (1993)], animated programming [ToonTalk, Kahn (1996)], 3-dimensional programming [Alice, Cooper, Dan & Pausch, (2000)], visual programming (RoboLab), and

graphical programming [Stagecast Creator, Smith & Cypher (1999); Icicle, Sheeham (2004)]. Given the wide range of different CPEs specifically developed for young learners, it is necessary to define which characteristics meet learners' programming needs and learning habits in science.

For the purposes of this paper two case studies of two different CPEs [Microworlds Logo (Papert, 1993) and Stagecast Creator (Smith & Cypher, 1999)] that use different program languages (formal textual language and graphical program language, respectively) are described and analyzed. The use of the two different CPEs was investigated in order to develop detailed and comparative descriptions of how students use these tools during scientific modelling. Traditional studies have failed to describe in detail how learners use different CPEs; they usually study the effects of programming on skill development or acquisition such as problem solving among learners using pre/post tests designs or describe the characteristics and capabilities of software (eg, Rader et al, 1997; Smith et al, 2000), without any descriptions of how students use them. For example, to program through typing instructions (in textual CPEs) might sound more difficult than assigning rules to objects (in graphical CPEs), especially given the lack of any scaffolding for programming. Running a program, however, by executing instructions is very different from executing a set of conditional rules (Ko, Aung, & Myers, 2005). Additionally, reading programs in a textual program language may seem harder, but one may wonder whether reading graphical representations of rules is any easier (Louca, 2005).

This paper is organized in four parts. The first part includes the theoretical framework, which (a) describes the analogy of computer programming as modelling,

highlighting the characteristics of the process of programming that are analogous to the process of scientific modelling, and (b) discusses the framework of different CPEs designed for young learners that use different types of program languages, to justify the selection of the study's two CPEs as representatives of the different program languages available for young learners. In the second part, the methodological framework is presented. In the third part, information on the findings of the study is detailed. Specifically, findings related with the (potential) role of CPEs in supporting (or constraining) student thinking and learning in science at the elementary level, and how student inquiry look within the context of working with CPEs in science, are reported. Finally, in the fourth part, the findings of the study are discussed.

## Modelling and Computer Programming

The process of model development and deployment may be compared to the process of writing and implementing a computer program (Louca, 2004). Most powerfully, it can be carried out through a computer program, when the program itself becomes the scientific model. In this way, the program language becomes the design medium for the scientific mode and the program (outcome) becomes a way of clearly articulating one's understanding about scientific phenomena. This has been the approach of a number of educators interested in computer-based modelling and science education (diSessa, Abelson, Ploger, 1991; Papaevripidou, Constantinou and Zacharia, in press; Redish & Wilson, 1993; Sherin, 1996; Sherin, diSessa, & Hammer, 1993; White & Fredriksen, 1998; Wilensky & Resnick, 1999).

Having to represent natural phenomena through models, students need to deconstruct their understanding of the particular physical mechanism into small

programmable pieces of knowledge in order to transform an idea in science into specific, technically precise program code (Louca, 2004). Therefore, programming (which is a principle tool in science as it provides a language for formal, technical precision and consistency) can be an alternative language for using in developing understanding in science (Sherin, 1996).

Programming has at least three important advantages over any other modelling-based approach. First, a program can be run on a computer and its results can be observed, allowing an iterative process of testing and debugging that may be more tangible and accessible for young learners than the iterative process of developing and deploying a scientific model expressed in other ways. Second, the code itself can be more easily read and explained than other principal tools for representation and communication of ideas in science. Lines of code can represent procedural instructions that depict relations among variables and objects that, given a sufficiently accessible language, the students can read and follow. For example, the programming languages that some CPEs use make the process of communicating ideas easier than the use of mathematical equations. Third, to write code that would create a simulation of a natural phenomenon, students are put in a context where they write code that would create a representation of the phenomenon (model/simulation) and thus, the idea of causal relationships between the phenomenon and its mechanisms can be more easily understood than when modelling with any other tool.

Additionally, the context of programming may help students overcome difficulties that they usually face in science learning. One of the difficulties that students confront is understanding the relationship between a scientific model and

"reality." However, in the context of writing a computer program to model/simulate the natural world, students tend easily to accept the notion that the program cannot reproduce everything, and thus the programmer must select which aspects of the world to represent and which to ignore (Sherin, diSessa & Hammer, 1993). In this way, the task of writing a computer program consists of creating, as Medawar (1987) described, an idealized "possible world".

The activity of programming may also bring the constraint of formal precision. Students learning science often struggle with scientific terms (eg, force, acceleration) that have everyday, context-dependent meanings. Science students need to learn new, more refined meanings of these terms, but, as importantly (and as difficult to accomplish), they also need to learn the practice of quantitative precision: For an idea to be useful in science, it should be made sufficiently precise in order to maintain consistent meaning across different contexts (Hammer & Elby, 2003). Developing models through computer programming can help students develop precise, operationally defined definitions of scientific terms.

## Program Languages

Currently there are numerous ways of classifying different CPEs (e.g., Singh & Chignell, 1992; Hogan & Tomas, 2001). One such classification is based on the type of the program language that CPEs use. At the one end of the spectrum lie textual program language systems (Papert, 1993) and at the other end of the spectrum lie graphical program language systems (Singh & Chignell, 1992).

The two CPEs used in this study were chosen following Singh & Chignell's (1992) classification based on the program language. One textual CPE [Microworlds

Logo (Papert, 1993)] and one graphical CPE [Stagecast Creator (Smith & Cypher, 1999)] were chosen to be investigated. The two CPEs were chosen among a large number of available programming environments that have been specifically designed for young learners based on previous experiences of using CPEs with young learners. Findings from previous studies (Louca *et al*, 2003; Papaevripidou, Constantinou and Zacharia, in press), showed that fifth and seventh grade students were capable of using Microworlds Logo and Stagecast Creator successfully for modelling natural phenomena. Table 1 below summarizes the most important features of the two CPEs used based on the program language they use, followed by a discussion of these features.

[Insert table 1 about here]

*Difference in the Program Languages.*

On the one end of the spectrum CPEs such as Microworlds Logo [a revised version of Logo (Papert, 1993)] use a textual program language as the medium for expressing relationships between objects. Using a textual program language provides the means for designing accurate mathematical models of the natural world. Despite its abilities to handle graphical representations and animation, Microworlds Logo's visual capabilities are limited to the outcome (simulation) and not the program, program language or the programming process. For instance, writing code that would simulate a character to move in constant velocity requires an entirely textual program as shown in figure 1.

[Insert figure 1 about here]

On the other end of the spectrum, CPEs such as Stagecast Creator are entirely based on a graphically represented program language. Programming in Stagecast is done by demonstration, using "click-and-drag" techniques (Smith and Cypher, 1999). During programming, the software records the user's actions storing them in a visual manner in a script consisting of visual "if-then rules" rules (Smith & Cypher, 1999): for a given situation, an action is determined. For instance code for simulating motion with constant velocity is as simple as defying a rule that would move a character one square per machine clock tick, as shown in figure 2.

[Insert figure 2 about here]

*Differences in the Program Strategies.*

Textual-based CPEs such as Microworlds Logo usually utilize procedural programming: the user types a sequence of instructions (written in the program language) for the turtle (character) to follow when the program is executed. On the other hand, programming in Stagecast Creator is object oriented: the user has to assign each character with its own rules that define its behaviours. Rules need not to be executed sequentially (although that is entirely possible) but are usually executed based on whether each rule's condition is met.

*Differences in the Representation of Objects.*

Stagecast Creator uses analogical representation: an object is represented in the same way in all different levels of the software (the program level, the outcome/simulation level etc). This way, the graphical environment that is used allows direct manipulation of the represented objects and easy assignment as well as direct review of the rules to each object (Smith & Cypher, 1999). On the other hand,

in Microworlds Logo objects are represented by an image in the output (simulation) window and by a given "name" in the program window. All instructions defining behaviours of all characters are placed in the same window, without any graphical differentiation between different characters. For this reason, even though many programming applications are considered object oriented (that is every object has its own identity and thus can be manipulated by the programmer independently), the way of creating, running and debugging program varies both in difficulty and complexity.

*Differences in the Representation of Physical Values*

In Stagecast Creator variables are clearly differentiated from the rest of the code: they are represented with boxes named after variables and are located below the list of rules of each object. Additionally, variables can be easily incorporated in the program by simply dragging them into a rule. Rules and variables are clearly differentiated and stored "behind" each object, where they can be reviewed any time, even during running a program: by double-clicking on an object, one can review its rules and variables.

On the other hand, variables in Microworlds are defined through written code, using particular program primitives, without any further differentiation between variables and the rest of the code: to locate a variable in the program the user needs to read through and identify the primitive creating or defining that variable. Additionally, if the programmer wants to have a way of reviewing the variable's value while a program runs, a different subroutine needs to be written for creating a visual representation of the variable's value. Lastly, changes in a particular variable are a

result of textual instructions (code) which once more is not differentiated between the variable itself.

## Purpose of the Study

Given the differences in the ways that CPEs have been developed to be used by young learners and the different characteristics that they have, it is necessary to define which characteristics are useful for student modellers/programmers in science. There is some literature that provides descriptions of the characteristics and capabilities of the CPEs, mostly derived from the process of developing and testing prototype systems (i.e., Smith, Cypher & Telser, 2000; Cypher & Smith, 1995; Rader, Brand & Lewis, 1997). However, there is scarcity of research studies that investigate how different characteristics of different CPEs could affect scientific modelling. It is equally important to learn how the limitations of the available CPEs affect the process of scientific modelling.

The purpose of the present study was to describe the ways that elementary school students use different CPEs to develop models of natural phenomena. In particular, this study comparatively describes the ways that two groups of fifth grade learners used two radically different CPEs, Microworlds Logo and Stagecast Creator, as modelling tools in science.

This study follows the qualitative research tradition for classroom-based studies focusing on students' activities and conversations during the process of developing models of natural phenomena, in the natural settings (context) in which modelling takes place, seeking to capture the classroom dynamic (Bogdan and Bilken, 1998). At the same time, large amounts of qualitative data that can derive from

studying the learning process can be analyzed (Bogdan and Bilken, 1998) to provide evidence that point to possible differences in the ways that these learners use the two CPEs to represent natural phenomena. Qualitative data are meant to be used as rich holistic descriptions of social phenomena and are analyzed in inductive ways (Bogdan and Bilken, 1998). In this interpretive case study we describe claims that seek to provide an emerging theoretical perspective grounded in the collected data (Bogdan and Bilken, 1998). Thus, comparisons of student activity and conversation patterns that we summarize below are not meant to provide conclusive claims about the effect of different characteristics of CPEs in the student use of the environment. Rather, we are concerned about "the process rather than simply with the outcomes (Bogdan and Bilken, 1998, p. 6)." Thus, although the importance of any learning outcomes (student-developed models for instance) is not underestimate, this study is concerned with the actual process of modelling and how learners used the two different kinds of CPEs.

This paper follows one of the current practices in science education research to develop rich, detailed case studies seeking to move beyond codings of classroom discourse, to describe in more detail what the classroom discourse looks and feels like in the real classroom (e.g. Kelly et al, 1998). Analysis through codings of student utterances can capture a lot of information regarding modelling but can also miss a lot of what is going on in the conversation. For this purpose we combine analysis through a coding scheme with a descriptive conversation analysis, aiming to provide detailed descriptions of two extended case studies.

## Methods

### *Participants*

This study took place at a suburban elementary school in Maryland, USA, and lasted four months. Thirty fifth-graders were randomly selected out of a group of fifty three students that volunteered for the study. However, out of the thirty participants only nineteen students remained until the end of the study. The eleven students that did not complete the study reported that they could not remain until the end of the study because of other extracurricular activities that they had to undertake.

Two afternoon computer/science clubs of fifth grade students were set up, one for each CPE. Initially, fifteen students were included in each club. After the leave of the eleven students, the Microworlds Logo Club was left with ten students and the Stagecast Creator Club was left with nine students. Students were primarily divided into the two clubs based on their indicated preference of the day they wanted to participate in the study. Each club was meeting with the first author once a week on a different day for one and a half hours.

The clubs were representative of the population/cultural diversity of the school, and they included five African-American students (three girls and two boys), two Latino students (one girl and one boy), one Chinese student (boy) and eleven Caucasian students (two girls and nine boys), a total of nineteen students, six of which were girls and thirteen were boys.

During the study, students worked in small groups of 2-3 members each. In all the cases, the distribution of students reflected both cultural and gender diversity among the groups. However, the most important factor for dividing students into

groups was to have group members who could work together successfully. To accomplish this, the opinions of the teachers of the school were taken into account.

All of the participants had some experience with computers, even though none of the participants had previously used any of the CPEs that were used in the study. The school has a computer lab, and a designated computer teacher with a teacher assistant. Students in the school regularly visit the computer lab, where the computer teacher in coordination with the students' regular teacher teach lessons in e.g., history, mathematics, social sciences, etc, which involve the use of computer software applications such as PowerPoint, Internet browser, etc.

*Study Parts*

The study was divided into two parts. The first part was devoted to learning the program language and some modelling procedures and the second part was devoted to developing models of natural phenomena with the use of the CPEs. The data analysis for this study is based on the data collected during the second part of the study.

*Study part I.*

The first part of this study took place during the first 6 meetings. Its purpose was to teach students how to use the CPE they were assigned to, and to introduce them to some modelling practices. More details about the first part of the study including the teaching approach and philosophy can be found elsewhere (Louca, 2004).

During the first sessions, students in the Stagecast Creator club familiarized themselves with the environment using the software tutorial that was previously demonstrated to be a successful tutor for this CPE (Papaevripidou, Constantinou and Zacharia, in press; Louca *et al*, 2003). The tutorial is an interactive environment presenting the capabilities of the CPE and showing the user how programming is done. After going through the tutorial, students were introduced to several examples of ready-made microworlds in order to investigate their structure, and practice their programming skills by altering features of the microworlds. This approach helps students to focus on the rules that create the simulation and to think of these as the mechanism of creating the simulation.

For the Microworlds Logo club, the teaching focus was on the program primitives and basic program structure. Teaching was also done through presenting students with simple pre-programmed microworlds, asking them to figure out how the behaviour of the characters was created and how to modify that behaviour. These activities provided students the opportunity to investigate the capabilities of the programming environment, and to develop an understanding about the function of programs in Microworlds Logo. Towards the end of part one, students were asked to develop their own simple programs. Due to the limited scaffolding provided by the software, students were allowed to use code from programs that they previously had seen or used.

*Study part II.*

During the second part of this study, each group developed a representation of a natural phenomenon. Prior to any work, group members collaboratively decided the

topic of their final project, to support different student preferences and likes. Students in each group spent a meeting brainstorming ideas about possible phenomena that could be modelled through the available CPEs. Each group selected a different phenomenon as shown in Table 2. The only limitation given to the students of each CPE club was that their topic should differ from the topic of the other groups. The purpose was to collect data from as many different science subject domains as possible. Due to the fact that the topic/phenomenon varied across groups within each CPE club, only findings that were common among all student groups using the same CPE are reported in this paper. The idea was to avoid reporting any findings that were topic/phenomenon depended and not CPE depended.

[Insert table 2 about here]

*Data Sources*

Three sources of data were used in this study. First, videotaped students' group work with Microworlds Logo and Stagecast Creator that includes both their interactions with peers and the first author. Second, discussions that the first author facilitated in whole class about the phenomena under study were videotaped and used as a source of data. After the end of the data collection period of the study, all audiotaped conversations were transcribed for subsequent analysis. Third, the first authors' reflective journals were also used as data sources, guiding the analyses of the first two data sources. These journals included reflective notes for individual lesson planning and lesson implementation, as well as observations regarding student modelling practices from each lesson.

*Data Analysis*

This is an interpretive case study (Creswell, 1988; Merriam, 1988) seeking to investigate students' work with two CPEs while developing models of natural phenomena. For analysis purposes, each CPE club was treated as a separate case. The case units for this study were the student groups working with each CPEs, and the subunits of each case were the weekly meetings with the students. For this purpose, analysis and presentation of findings were based on all four groups from each CPE club (case study units) following their work in detail for almost two months of meetings (case study sub-units). After the separate analysis of each case (CPE), the findings from the two different cases (one for Stagecast Creator and one for Microworlds Logo) were compared, to isolate their differences and similarities in terms of students' (a) activities and (b) conversations.

For the data analysis, two different types of analysis were used: contextual inquiry and analysis of student conversation. Contextual inquiry was used in conjunction with analysis of student conversation in order to gain better insight in students' activity and conversation patterns while working with CPEs and to triangulate findings (Stake, 2000). Triangulation helps to support claims, by using two different analyses from two different theoretical perspectives to point to similar findings. This combination of findings provides a better, more detailed picture. Snippets of student activities and conversation that are presented in this paper were selected as examples to support the claims from this study. Combination of contextual inquiry and analysis of conversation were used to develop two detailed case studies. That work is presented in detail elsewhere (Louca, 2004). What follows is a detail description of the two types of analysis used in this study.

*Contextual inquiry.*

Video and conversational data of children's work with CPEs were analyzed using a modified version of Contextual Inquiry (Druin, et al, 1999). Contextual Inquiry is a method of collecting and analyzing data of children's activities and conversations, and it involves the analysis of student work in a particular macro-context such as using technology and computer media.

Transcripts of videotaped conversations were separately coded for (a) activity and (b) communication patterns. Every student utterance was placed in a different cell [decision adopted from a study investigating student interactions while working with Stagecast Creator (Underwood, et al, 1996)]. Moments of silence were represented in separate cells, and were added after reviewing the videos. Codes for activity and conversation patterns emerged from the data following open coding from grounded research methodology (Strauss & Corbin, 1998) trying to capture student's activities and conversations during their work with the CPEs. Categories were developed during the process of coding, and after the list of codes was finalized, all coded transcripts were reviewed once more, to check for consistency in the applied coding. Coding was then repeated by another coder (Cohen's Kappa = 0.82), who did not have access to first analysis. Differences in the assigned codes were resolved through discussion.

Students' activity and conversation patterns were then separately presented in time-line graphs, following the approach of Schoenfeld (1989). The x axis of the graph represents utterance number from transcripts and the y axis represents categories of activity patterns (see figure 6 for an example) and conversation patterns

(see figure 5 for an example). Overall, for each student group, two graphs were produced.

After converting all data into graphs, graphs from units of the same case (groups from the same CPE club) were compared to isolate similarities in the combinations of the patterns of students' activities and conversations. From this comparison activity and conversation types emerged, based on combinations of codes that were similar among all analyzed groups for each CPE. In this paper, only activity or conversation types that were observed in all groups' data are reported.

The presentation of activity and conversation patterns follows three distinct phases of student work, during which patterns across different groups of students working with the same CPE shared similarities. Those phases also emerged from the data, since the activity and conversation patterns changed dramatically from one phase to the other leading to the decision of grouping them and presenting them as follows: (1) approaches to planning, started from the moment that students decided what phenomenon to work on and ended when students started working on their computers, (2) writing and debugging code, started from the moment that students started working on their computers until they had their first successful program/model running, and (3) approaches of using the code of their programs as phenomenon representations, started after students had their first successful program/model until the end of their modifications to their program/model.

*Analysis of student conversation.*

Analysis of student conversation was the second type of analysis that was used in this study. It is a multidisciplinary approach of analyzing text such as transcribed

student conversation, as a gateway to student thinking and experience. Patterns of students' conversations (revealed by contextual inquiry) and analysis of student conversation are presented together to triangulate findings. Analysis of student conversation provides in detail the particular context in which students' work (activities and conversations) took place. Student conversations happened in a variety of situations (micro-contexts) that contextual inquiry does not account for, such as while students were away from the computers, while programming, while debugging, while changing how their simulation looked etc. Therefore, the purpose of the analysis of student conversation was to map possible relations between the conversations and the context in which they happened. For this purpose, in the examples below raw transcript data of students' conversations are provided along with the possible explanations of what kind of thinking is taking place in the conversation, both as one possible gateway in student thinking.

Analysis of student conversation is multidisciplinary because it uses research techniques and approaches originated from linguistics, educational psychology and educational research (Edwards & Mercer, 1995). The kind of analysis of student conversation that was used in this study also follows examples of such approaches for analyzing student conversation in science and mathematics (e.g. Ball, 1993; Gallas, 1995).

Analysis of student conversation is different from discourse analysis (Sinclair & Coulthard, 1975), because it does not seek to reveal the structure of the talk. It is rather focused on the context in which the conversation takes place and in the content of the conversation (Edwards & Mercer, 1995). For this reason, analysis of student

conversation does not follow a process of coding text with particular codes, but rather it provides detailed descriptions and possible interpretations of the conversation, that are meant to be read in parallel with the transcript. In this sense this follows the research approach of educational research (Edwards & Mercer, 1995), which seeks to develop a sense of what takes place in the classroom in an effort to map possible relations between the learning processes and the discourse.

## Findings

This section presents a comparison of student activity and conversation patterns when using one of the two CPEs with respect to three distinct phases of student work that emerged from the contextual inquiry data analysis that include their approaches to (i) planning, (ii) writing and debugging code and (iii) using code to represent the phenomenon under study (see Table 3). The reason for creating these three phases was the fact that the activity and conversation patterns of each one of these phases was dramatically different from the others. However, the activity and conversation patterns across different groups of students working with the same CPE shared similarities within each one of these phases.

Assertions for each student club (Microworlds and Stagecast) are supported by the presentation of contextual inquiry data and excerpts of conversation analysis, following the work of two student groups (one per CPE club) that were chosen randomly. The findings that are presented are those that were observed across all groups within a CPE club. Thus, there is no need to report the corresponding prevalence of each one of these findings within a CPE club or across the CPE clubs. To maintain consistency in the presentation of the findings, we chose to use examples

of data from the same group of students. However, in one case we included data from an additional Stagecast Creator group, to highlight some of the difficulties that students encountered with the third phase of their work.

Claims that are provided below are supported by findings from contextual inquiry (presented in the separate timeline graphs: one for activities and one for conversation patterns), short excerpts from student conversation accompanied with their conversational analysis, and examples from student models. References to student models below differentiate between their code and the model/simulation that is a result of executing that code. For the purposes of this paper, we treat models to include both the code (that causes the model/simulation) and the resulted model/simulation.

*Different Approaches to Planning*

The data analysis has shown that students working with different CPEs, as far as their approaches to *planning* are concerned, differ considerably. At the outset of their work, even though, in both clubs the students started *planning* their work by breaking down their ideas, Microworlds Logo students grouped together parts of the phenomenon based on the behaviour of the objects, whereas, Stagecast Creator students broke down their ideas based on the sequence of the events in their story line: what would happen first, second, third and so on. It was almost like the Stagecast Creator students described a movie, talking about each scene one by one. In this sense, Microworlds students made plans based on the structure of their programs whereas Stagecast students focused on the overall story, talking about sequential scenes of that story.

While planning their work with Microworlds Logo, *students talked about the structure of their programs*. They broke down the phenomenon under study into small pieces, based on the behaviour(s) of the objects. Parts of the program in which an object had a similar behaviour were grouped together (e.g., the object moved in the same direction). Although students' grouping criterion was the similarity in the object's behaviour (reflecting some science content), their conversations were very technical, mostly about the structure of students' programs (reflecting a programming perspective). For example, students talked about how many subroutines (small independent programs (routines) that can be grouped together in a larger program) they would have and why, and how different subroutines represented different parts of the phenomenon.

In one group for instance, Joe and Samir (Microworlds Logo Group 1) talked about their program that would create a simulation representing how an arrow travels in the air. They talked about writing several small programs, each one to represent a different part of the arrow's trajectory, following their experiences with how the phenomenon looks: one program (subroutine) will correspond to the upward motion of the arrow, another one to the horizontal motion and a third subroutine to the arrow's downward motion (see figure 3). Following this idea, Samir got into more details about their program, and talked about writing code that would change the direction of the motion of the arrow (angle). However, he did not talk about details of the actual code they would use (how much the angle would change in each program, or how much forward would the arrow move before changing angle). This is reflected in the following small excerpt.

**Joe**: ok, what we're gonna have is an archer who's, you could do this later <gestures showing changing the angle of the shooting arrow>, we can make one big program, or we can make a bunch of little programs to make it, cause that wouldn't be as realistic because the arrow would go hum, hum, hum. <gestures showing the upward, the horizontal and the downward motion of the arrow>

[…]

**Samir**: …we had this really good idea that we want to share with the two programs. What we can do is that we're thinking that we can make a little …, we can make something to adjust, instead of the wait, or the fd, we can adjust the angle, so that makes more and more and more and more, and then we can stop it when it's about to go down, and then make a program, minus-ing it…

*Excerpt 1*

As the conversation continued, Joe talked about an alternative idea, to write a program that would "do" half an oval, to resemble the trajectory of the arrow (see figure 4), again in the absence of any details about the actual code that they would use. Unlike their first idea, the essence of creating an oval shape is a program that describes the physical mechanism of changing the arrow's direction, even though students were not talking explicitly about that. Without realizing, students entered a conversation about representing a mechanism that could account for the change of the direction of the velocity's vector of the arrow. While focused on constructing a model/simulation of the phenomenon that would look like having an oval-shaped

trajectory, Joe's program included, most probably unconsciously, a representation of the mechanism that changes the arrow's direction.

[Insert figure 3 & figure 4 about here]

On the other hand, students working with Stagecast Creator planned their work by *talking about the scenario they were about to program*. They tended to focus on the overall story of the phenomenon under study and talked about what their simulation would look like. Details of the scenario that was underlying their plans seemed to be important for students. In one Stagecast Creator group for instance, Annie and Bryan (Stagecast Creator Group 1) talked about creating a balloon shoot-out game. It was a game about shooting down helium balloons that travel at different speeds. They talked about how balloons would be shot down, about scoring and the purpose of the game, providing lots of details about their game scenario. Conversations about their designs at this point of their work were presenting descriptions of sequences of events that students planned show through their simulation. This is reflected in the following excerpt.

**Annie**: we, the name that we end up deciding to call our game balloon shoot-out and the idea is that are series of colored balloons and the different colors make them bigger. And…

**Bryan**: for example like the less point there, the bigger balloons are.

**Annie**: yea, like gold is to be tiny but it's worth 50 points. And there's different ones with <inaudible> but gray we got a gray balloon and it'll be a wipe-out and like the points will be gone. And through deciding may, we don't know

how many levels we're gonna make, and we decided that 3 wipe-outs is the end of that level. And if you are like level 2, you have to go down to go to the bottom.

*Excerpt 2*

Like students working with Microworlds Logo, *Stagecast Creator students were also breaking down their ideas*, using, however, a different criterion. While Microworlds Logo students were grouping together parts of the phenomenon based on the behaviour of the objects, Stagecast Creator students were breaking down their ideas based on the sequence of the events in their story line: what would happen first, second, third and so on. It was almost like students were describing a movie, talking about each scene one by one.

*Differences in Approaches to Writing and Debugging Code*

When they started working on their computers, students using different CPEs were seen to operate in different states of mind. Students working with Microworlds Logo maintained an "authorship" relationship with Microworlds Logo trying to write programs that execute and then shifted their attention, focusing on getting a model/simulation that would look realistic. On the other hand, the act of developing rules, caused students working with Stagecast Creator to shift into having an authorship relationship in an effort to write a program that would show their story, translating sequential scenes of their scenario into programmable rules in Stagecast Creator. Microworlds student conversations were still technical and much more limited than the Stagecast student conversations.

[Insert figure 5 about here]

During the early phases of their work with computers, students working with Microworlds Logo set as a goal the development of programs that *would run*. In their group, Samir (Microworlds Logo Group 1) for instance, undertook the role of the "typist" and typed their program while Joe simply watched. During typing, students' conversations were limited, and happened only in cases of mistyped primitives or to make sure that they had them right, thus making the conversations technical. Figure 5 reports on contextual inquiry analysis, presenting a segment of Joe's and Samir's conversations during typing and debugging. Most of the time only one student was at the computer (represented by the "one leaves group" category) or students were not having any conversations (category "silence"). For the rest of the time, their conversations were still technical, either giving each other directions about what primitives to use in their programs (categories "how to program (with code)", and "give direction"), and talking about what they were doing on the computer (categories "what are you doing?" and "now I click here").

Joe's and Samir's first programs also reflected the authorship relationship with the CPE: a program at this point was acceptable if it could execute. Their program resulted a model/simulation that although looked fine (showing an arrow moving in the air), it was running on code that was very different from what they had talked about before, indicating that their goal was to get a program that runs successfully, despite their prior planning.

[Insert figure 6 about here]

The act of formal programming may have been responsible for both the limited conversations and the structure of their first program. Programming in

Microworlds Logo was mostly typing code and consequently students' first programs usually consisted of a single subroutine that included a number of instructions of what the object would do sequentially. For instance, figure 6 presents Joe's and Samir's first program which clearly demonstrates two things. First, students seemed to had started working on the ideas they discussed during planning – that is why they had created a variable for the angle of the motion and set its initial value at 45 degrees. Second, the following two lines of their program seem to neglect the angle variable, possibly reflecting the difficulty that students had to get their program to run. As screen capture data indicate, their initial efforts to write a program that would successfully change the value of the variable failed, and students simply focused on getting their program to run and create what they thought it looked fine as a representation of the phenomenon. Once again, these instructions seemed to *have* an underlying mechanism that was causing the change in direction of the motion, even though it was not represented in the program.

[Insert figure 7 about here]

Unlike typing, debugging was a process of going back and forth, from the program window to the simulation window and vice versa. The version of Microworlds Logo that was used in this study cannot display the program and the simulation at the same time, and users need to switch between the two windows. Figure 7 represents coded activity patterns (contextual inquiry analysis) from Joe and Samir's group. Up to about utterance no. 90 (see figure 7) Samir was typing their program and after that they were switching between the program and simulation window, run their program, read some feedback and then make changes in their code.

Conversations during that time were also limited. Students focused on getting the model/simulation to run. In this sense, the code was used by students as a tool, but in a rather non-productive way, at least for scientific modelling, since the goal was to get a model/simulation that run (not a simulation that represented a phenomenon). However, having a program that runs successfully is probably a requirement for a conversation about how the particular program represents a natural phenomenon.

When their program was bug-free, students working with Microworlds Logo started talking about the resulting model/simulation and *their focus shifted to getting a simulation that looked "realistic."* Their conversations during this time were about what possible changes could be made to improve their simulation. While the code was used once more as a tool to modify their programs to result in better simulations, the structure of their program was not a concern any more, but rather, students were making *any* necessary changes to improve their simulation. In several cases changes departed significantly from their plans, indicating that the students were focused entirely on how the simulation looked and not on how to represent the phenomenon they were studying.

This shift in student focus is reflected in the following excerpt (analyzed by analysis of student conversation), that starts after Joe and Samir had successfully debugged their first program. Having a program that ran, students started talking about how the simulation looked, a clear shift of interest from the code to the simulation. In the conversation below, students were focused on changing their program in such a way that the simulation would look better. They were concerned

about how their simulation looked; they made a few changes to their code, mostly to numbers, (amounts of forward, waits, directions etc) and then they tried them out.

**Samir:** that was way too many repeats, that was….

**Joe:** do it again, do it again!

<laughter>

**Samir**: we have way too many repeats.

<silence while clicking/typing/looking on the screen>

**Samir:** I know what we have to do, we have to do seth 90.

<silence while clicking/typing/looking on the screen> - Joe left the group

**Samir:** I got it!

**Richard:** hey, you got it,

**Samir:** well, it looks better now.

**Richard:** hey Joe look what Samir got.

**Samir:** heeeeeey!, I got it somewhat!

**Richard:** you got it down, look, Joe, look!

<silence while clicking/typing/looking on the screen>

*Excerpt 3*

At this point of their work, their simulation looked fine, mostly what the students had talked about before (excerpt 1): the arrow travelled first upwards, then horizontal for a while and then downwards, stopping when reaching the target. Their code, however, did not reflect any of their ideas about having either small programs that account for each "phase" of the arrow or having a program that caused an oval-shaped trajectory. Their first program may had created a similar simulation with what they were thinking during planning, but the code of their programs was significantly different. It almost seemed that Joe and Samir started thinking in at least two useful ways, but they sacrificed them for how their simulation looked. When students were asked about this, Samir indicated that he was aware that their program was different from their plans. However, he indicated that after trying their plans, their first program was "the only way that worked out." And he continued: "I tried to make it the way we had it [in their plans] and it did like this …" gesturing that he was not pleased by how the simulation looked like.

On the other hand, the students working with Stagecast Creator moved from planning to programming. Their work and conversations reflected an effort to *translate the details of their scenario into programmable rules*. For instance, Annie and Bryan (Stagecast Creator Group 1) talked about how to make balloons in Stagecast Creator move in different speeds. Unlike students working with Microworlds Logo (who were typing and debugging code with limited conversations), programming with Stagecast Creator was an interactive process of both work and conversations, probably due to the dynamic process of programming. Figure 8 presents students' conversation patterns during their work with Stagecast Creator

(contextual inquiry), which presents a radically different situation from what happened with Microworlds Logo students.

[Insert figure 8 about here]

Stagecast Creator students did not have any clear plans about how exactly to create rules with Stagecast Creator. Rather, in a collaborative effort they translated the details of their story into programmable rules, as it is represented in the "tell story with no code" and "how to program" coding categories in Figure 8. *Students' focus was on creating a simulation that would show their story*. For this reason, they wrote, deleted, and re-wrote rules in an effort to create a simulation of their story. The code was solely used to create a simulation that would demonstrate their story, as a succession of events, one following another. In this sense, students maintained the focus on the overall story, which they had during planning, to tell a story as a succession of events. An example of student activity patterns during this phase of their work is presented in figure 9, followed by a small conversation excerpt from Annie and Bryan's group. Figure 9 represents contextual inquiry findings and shows that students typed some code, tried it out, write some more code, and tried that out, too.

[Insert figure 9 about here]

**Annie**: yea, but we want different colors [for the balloons].

**Bryan**: can we make up a green.

**Annie**: and gray and yellow and gold.

**Bryan**: the slowest is red, right?

**Annie**: yes. The slowest is red.

<silence while clicking/typing/looking on the screen>

**Annie**: all right. Let's make it move, um 2 boxes, at a time

**Bryan**: like, why not 1?

**Annie**: cause that would be so easy to shoot.

<silence while clicking/typing/looking on the screen>

**Annie**: done. Now let's play. Right.

<silence while clicking/typing/looking on the screen>

**Bryan**: No, that should be the gold.

**Annie**: yea. That's too fast.

*Excerpt 4*

Students' early programs in Stagecast Creator were successions of rules representing "scenes" from their story, which Stagecast Creator was running in sequence. For instance instead of having one rule that would increase a balloon's increasing speed, students had a rule for moving one square per software cycle[1], then a rule for moving two squares, and then a rule for moving three squares (see figure 10). Unlike students working with Microworlds Logo, Stagecast Creator students wrote each rule and tried it immediately before moving to the next. If a rule did not have the expected results, students did not spend any time figuring out what was wrong (debugging). Rather, they deleted the rule and created a new one. In figure 10,

contextual inquiry findings show that students wrote some code, then they tried it, if they had to change it they deleted it, typed some new code, tried it, then typed some more, tried it and so on.

[Insert figure 10 about here]

In addition, debugging was only in the form of deleting rules (that had unwanted effects) or changing the rule sequence. Syntactical bugs are almost impossible in Stagecast Creator programs because the software provides a lot of scaffolding for creating new rules. The user has simply to demonstrate to the system the desired behaviour or fill in blanks about the conditions of each rule and the effect of the rule (desired behaviour).

*Differences in Using Code as a Representation of the Phenomenon*

There was a third shift in student focus during their work with CPEs that was mostly observed with students using Microworlds Logo. This was a shift to use CPEs as modelling media. For this shift to occur, students working with Microworlds Logo had to start reading their code, instead of simply running it to see the resulted simulation.

When students working with Microworlds Logo read and talked about the code in their programs, they saw it as a representation of the phenomenon (e.g., representation of the behaviour of the objects in the model/simulation). In other words, they used the code to talk about how the phenomenon occurred, rather than seeing the code as a tool that created a model/simulation. Students in this mode of work saw the code as the representation of the phenomenon itself. Conversations about the representation of the phenomena in Microworlds Logo code resulted in

iterations of model refinements, mostly in an effort to include a mechanism that would show how the phenomenon happened. In this sense, students moved from programs that were simple descriptions of the phenomena, to creating causal models that caused the phenomenon.

When Nick, for instance, saw Joe and Samir's simulation about the moving arrow, he indicated that it looked fine ("That [the simulation] looks ok"). As soon as he saw their code, he indicated his puzzlement:

**Nick:** no, actually this program isn't what an arrow does! But anyway. An arrow actually, wait, sorry, but…that's what a rock does. What the program is doing that would what a rock does. This is what an arrow does. An arrow drops just like a gun bullet does! A gun, like when you shoot a gun, the bullet would drop.

*Excerpt 5*

For Nick to identify that "this is not what an arrow does" required him to see in the code something different than what he expected to see. Even though the simulation looked "fine" for him (as he indicated a few minutes ago), the code did not. This possibly suggests that what Nick read in the code a causal mechanism that was different from the one he expected to read.

Nick's comment sparked a new kind of conversation, which had two unique characteristics. First, the focus of the conversation was now on the code itself. Nick indicated that the first part of their program (that resulted in the upward motion of the arrow in the air) was not what it should have been. Things that are shot in a 0-degree

angle (like the arrow here as indicated by the program) continue to move straight and "drop a little" (possibly implying that in no way do they go upwards). Secondly, Nick backed up his idea by making reference to experiences he had from other situations that could possibly be clearer; such as the case of a gun bullet, as indicated in the above excerpt.

On the other hand, to talk about causal mechanisms, students working with Stagecast Creator had to shift their focus from showing a story through the simulation, to talking about the different concepts (variables) in the story that affected object behaviour such as food, energy or speed or acceleration. Like students working with Microworlds Logo, students working with Stagecast Creator had conversations about how things happen in their simulation, utilizing the scaffolding of Stagecast Creator to talk about the causal mechanism of the phenomenon. For instance, while working on their balloon game, Annie and Bryan (Stagecast Creator Group 1) entered a discussion about representing a general mechanism that would cause their balloons to move according to the amount of helium inside them. Their first rules were simply descriptive of balloons' behaviour, but they soon realized that they could develop a set of rules that can actually "read" the amount of helium (variable) in each balloon and make it move accordingly.

The scaffolding that Stagecast Creator provides for rule creation seemed to support conversations about the mechanism; the difficulty, however, was to start such a conversation. Students seemed to see their work with Stagecast Creator as developing games that would include natural phenomena, rather than representing the phenomena *per se*, and they focused on the overall story, which seemed to get in the

way of scientific modelling. This was partly because to have a modelling conversation students had to talk about what caused the changes in the behaviours of the objects in their game scenario. Creating rules for simulating particular behaviours is much easier than creating general rules that cause changes in the behaviours (instead of multiple rules causing multiple behaviours).

The following episode from Zen and Seth's group (Stagecast Creator Group 3) illustrates the difficulty for focusing on causality, possibly due to the student's focus on the overall story. In this excerpt Zen and Seth started developing a prey-predator model/simulation in a lake with fishes. They wanted to create a mechanism that could account for, and limit the number of fishes that sharks ate. The idea was simple: the fish should have a limit of how many fishes they eat, and also how they "use up" the fishes they ate so that they "get hungry" again. Seth was finding the solution he was thinking amusing: whatever goes in the stomach, has to come out, one way or another.

**Zen**: we want the limit [of the number of fishes it can eat] to be 3. And then, after 3 it will stop for a few seconds and then, I am not sure where it would come out, but then the, like bone figures would come down to the ground.

**Seth**: it will blast and it will be bones in it.

*Excerpt 6*

The conversation continued about a succession of events in a story: one feels hungry, she eats, and she is then full and then there is something coming out of your body so that you can become hungry again. At the same time Zen and Seth were not

specific about how all these happen – possibly because this was not a requirement for telling the story.

**Seth**: when you're hungry you're…

**Zen**: your energy is lower.

**Seth**: because you don't have any to digest.

**Zen**: and then, um, the stuff from the food going to go to your blood stream and make your energy go up

**Teacher**: how does your energy go down?

**Seth**: you're hungry…

**Teacher**: how do you get hungry though?

**Seth**: yea seriously, how does it [your stomach] get empty?

*Excerpt 7*

As the conversation continued, Zen suggested that when you eat, the food gets in your stomach and then somehow some energy gets into your blood stream, without providing any details That was a step towards a mechanism that could account for the phenomenon, which probably helped Seth to indicate with puzzlement, in a similar mode of thinking,: "yea seriously, how does it [your stomach] get empty?"

Even though the conversation was about hunger and despite their willingness to develop rules to account for eating limits, using Stagecast Creator to tell the story of hunger was evidently getting into their way of thinking about the mechanism that

was causing hunger. Partly, this was because there was not a clear or even needed connection between the successive events, other than their sequence.

For the conversation to become productive, students should have talked about the story of the energy, rather than the story of hunger. The story of hunger is simply a description of the phenomenon, whereas the story of energy is a different kind of story. It is the story about a particular concept (variable) as well as the story of the causal mechanism of the phenomenon. This second story is much easier to program, because telling "the story of the energy," can be more productive. Different behaviours of "energy" (e.g., consumption, enrichment etc) are programmable pieces by themselves, and can be represented in rules. Programming the overall story requires identification of the objects, their behaviours and what causes those behaviours before moving to any rule creation.

That is why, when another student from another group brought up the analogy of the gas consumption seemed to helped Zen and Seth start thinking about food digestion. Food can be like gas, Zen indicated (excerpt 8), which is consumed when the fish moves. Students stopped talking about events of hunger, and started talking about events that can happen to "energy". They were most likely in the same "state of mind," that of telling a story, but this time their story was about energy, how it is regulated and how it is consumed. In a way story-telling mode was used in a more productive way for science, when minutes earlier it was used in a non-productive way.

**Jeremy:** it'll like, this is like a car using up gas.

**Zen:** the gas goes into the engine and used up for moving.

**Seth:** yea! So, if I have at the beginning of my journey I have 100 gas, at the end of my journey I will have like I don't know 30, depending on the distance that I have traveled.

**Zen:** Yea. So, food is like the fuel, and like the fish is like the car. I mean it uses some of the food to keep going and then rest stays [in its stomach] until all is used up! Only then, does the fish become hungry again…

*Excerpt 8*

[insert table 3 about here]

## Discussion

This section discusses the findings of the study based upon the differences, in student approaches to planning, writing and debugging code and using code as a representation of the phenomenon, that appear to exist between the two CPE clubs when modelling natural phenomena. Possible implications for modelling through programming in science are also addressed, with respect to the two different CPEs used in this study.

### *Different Approaches to Planning*

In both Microworlds Logo and Stagecast Creator clubs the conversations about *the structure of their program* were starting points of productive conversations about science. For students working with Microworlds Logo, breaking down phenomena into programmable pieces, based on the shared behaviours of objects, was helping students think about possible similarities and differences between the different parts of phenomena and that subsequently helped them focus on the

behaviour of individual objects. The subsequent development of programs was based on the differences of the objects' behaviours, even though students were still far from representing what was causing those changes in the objects' behaviours.

By talking about the structure of their programs, without any particular prompt, Microworlds Logo students got into a discussion about breaking the phenomenon represented into pieces based on the differences (or similarities) of the objects' behaviours. Modelling can be thought of as the representation of the causal mechanisms of the phenomenon, and the first step for identifying that mechanism is to isolate the differences in the object behaviours and then identify what causes those changes. Microworlds Logo helped students work towards this goal: after identifying differences in the behaviours, students started writing code that would cause that behaviours and thus they started developing a causal representation of the phenomenon in the program language.

On the other hand, talking about representing scenes of a scenario, like what Stagecast Creator students did, is probably not a productive conversation for modelling in science. Students working with Stagecast Creator talked about the physical system (overall story line) that they wanted to represent through their programs, and about the system characteristics and the system changes. However, system behaviours and changes are caused by object behaviours, and more importantly modelling a physical system requires representing objects' behaviours that would subsequently cause system behaviours (Colella, Klopfer, & Resnick, 2000). In a sense, one cannot model a system behaviour, simply because a system is not an object, but consists of a number of objects. Instead, an individual needs to think

about the system's objects and the objects' behaviour that cause system behaviours and system changes.

This seems to create a paradox: although Stagecast Creator students were using an object-oriented medium, they planned their work thinking about the system and not about the individual objects. A model/simulation in Stagecast Creator can be easily seen as a presentation of a story, even though what is going on in the story is based on how the story's characters act. In this sense, talking about the system and the system changes was not a productive conversation for modelling, because students were then required to translate those ideas into rules about the system's object behaviours. Unlike Stagecast Creator students, students working with Microworlds Logo were planning their work by talking about their program's structure even though they were using a CPE that follows sequential programming.

Right from the beginning of their work, students working with different CPEs were engaged in different "states of mind." Students working with Microworlds Logo had an *authorship relationship* with Microworlds Logo: their work and conversations reflected an effort to write programs and were focused on their program's structure. Students in Stagecast Creator, however, seemed to have a *visual relationship* with Stagecast Creator, operating within a mode of *creating models/simulations* that would show their story.

It only makes sense that students would take these different approaches to planning, if one thinks about it, because of the difference in the features of the CPEs that they used. Microworlds Logo is a textual-based, open-ended environment and does not provide any scaffolding for writing programs, which adds the difficulty of

having to write a program that can run successfully. On the other hand Stagecast Creator uses programming-by-demonstration interface, including a lot of scaffolding for rule creation (Kiper, et al, 1997), which makes the process of programming simple situations very easy. Assigning behaviour to objects is as easy as recording a desired behaviour, and thus students do not have to be concerned with the details of programming. Since their purpose was to create a model/simulation, they were concerned with the details of their story line that would be represented in the model/simulation. After all, telling a story well, does not necessarily include how that story is caused and what is the objects' role in creating a story.

*Differences in Approaches to Writing and Debugging Code*

During writing and debugging code, students using different CPEs in the study were seen once more to operate through different states of mind. Students working with Microworlds Logo seemed to have shifted their focus to having a visual relationship with Microworlds Logo, focusing on getting a simulation that would look realistic. On the other hand, the act of developing rules, caused students working with Stagecast Creator to shift into having an authorship relationship in an effort to write a program that would show their story, similar to what students working with Microworlds Logo had during writing and debugging their code.

In all, data from this domain (writing and debugging code), possibly indicate that the act of developing programs from scratch does not seem to be productive for modelling. In both cases, students were not concerned with the representation of the phenomenon and its causal mechanism in the code, for which at least Microworlds Logo students were concerned during planning their work. Writing new code or

creating new rules has the added difficultly of getting them right, either to be bugs-free or to have the desired effect on the object's behaviour. The only way to check this was to watch the model that their program resulted in. In the case of Microworlds Logo students, they shifted their focus from the structure of their program to how their simulation looked.

In the case of Stagecast Creator students, the process of programming was a rather vocal stage in students' work, with students talking about how to translate particular ideas into rules. During that time, students realized that in order to write programs, the scenes of the desired story had to be translated into objects' behaviours and changes of those behaviours during those scenes, adding the difficulty of translating a story of a system into rules about individual object behaviours.

The two above problems, however, (1) type and debug code to get a program that runs in Microworlds Logo, and (2) translate a story line about what happens into individual rules in Stagecast Creator are different kinds of problems. The first is adding more work to students; before they can have a conversation about a model/simulation and its program they need to get the model/simulation to run. The second problem however, is more of a perspective issue: had students in the study seen planning as talking about the behaviours of individual objects, then they would not have any trouble during this phase of their work. Of course, there are possible ways that can make this part of students' work easier, such as providing students working with Microworlds Logo with some scaffolding for code writing. It is also possible that the more experienced with programming students get, the quicker they would move through this part of their work. The latter can possibly apply for students working with Stagecast Creator, too, whose experience with programming in

Stagecast Creator may help them to think from the beginning in terms of the objects' behaviours. On the other hand, there are physical phenomena that are based on the behaviour of a single object (e.g., a ball falling), in which case thinking about the story of the overall system is productive for modelling, because the overall story line of the system is the story of the object and its behaviour.

*Differences in Using Code as a Representation of the Phenomenon*

When students working with Microworlds started having conversations about how the phenomenon is represented in code, they still talked about parts of their program, but their focus was on the behaviours of objects in their model, and how they were reflected in the code. They talked about how the model *should* look and how individual characters *would* behave. The shift in their thinking was most probably due to their reading of the code – the model itself was not helpful in helping them think about what the arrow was doing. Rather than writing code from scratch, having to simply modify code, helped students focus on the science that their program represented (rather than on the code itself or on how the model looked).

For students working with Stagecast Creator, telling "the story of an agent" (e.g., energy, velocity, acceleration etc.), a particular concept that could be represented by a program variable, was very different from telling the story of a system that consists of multiple characters. The story of the system is a succession of events, which in order to be programmed one needs to identify objects' behaviours that cause them and program those particular behaviours. Telling "the story of a causal agent" was using a "story-telling" approach in a productive way, putting together the pieces of the story about energy. This was productive, partly because

different behaviours of "energy" for instance (e.g., consumption, enrichment etc) can easily be different programmable pieces that can be represented in rules. In this sense, the story was (partly) the causal mechanism of a living fish, and thinking in that way was productive for developing representations of that mechanism.

## Conclusions

This study investigated and comparatively described the ways that 19 fifth graders used two different CPEs for scientific modelling. Throughout this study, the research focus was neither on the students' knowledge (or ideas) nor on the knowledge that they gained. Rather, the focus was specifically on the ways that students constructed models of natural phenomena with computer-based programming tools designed for young learners and the types of programs that they created. In this sense, the focus was on students' activity and conversation patterns, as well as on the way they viewed the programming process as expressed through their work.

The findings of this study appeared to show that differences in the program language influence the "mode of work" that learners enter when using the CPEs, pushing their learning experiences into different directions with an effect on both the programming and modelling processes. Consequently, the type of the programming language has implications on (a) the programming process: textual language systems are more open-ended environments, enabling users to create many kinds of routines with limited scaffolding, whereas graphical language systems restrict users to pre-defined scaffolding for creating programs, and (b) the modelling process: Microworlds Logo which is a textual language system seemed to more easily trigger causal accounts of natural phenomena whereas Stagecast creator which is a graphical

language system seemed to better support narrative accounts. These findings have implications for teaching and learning in science, because this study primarily involved documentation and analysis of actual student work with computers as tools for learning.

Although findings are not meant for generalization in the student population, they could serve as a basis for further investigations into how learners use programming as a modelling tool in science in two ways. These findings may help re-define the questions that teachers ask when using CPEs with young learners. It was evident that students in this study entered a particular "mode of work" depending on the kind of the programming language of the CPE they used. Therefore, teaching purposes and decisions, student abilities and learning styles should be viewed through this lens. Additionally, future research may investigate whether the different ways that learners use CPEs such as the ones used in this study, and the differences in these CPEs can lead to different knowledge representations of physical phenomena and different types of models.

Finally, findings suggest that despite the overall differences in student work, each CPE had design features that were productive and helpful for students in particular contexts of their work. While a recommendation for how a good CPE should look is not applicable for this study (partly due to the study design and analyses used), software designers might use findings to determine which combination of characteristics to include in future software, depending on their purpose. Future researchers may also find it useful to investigate in more details (a) how students use, for instance, debugging and code in various CPEs and (b) the

particular activity contexts that can support productive science conversations during modelling.

References

Ball, L., D. (1993). With an eye on the mathematical horizon: dilemmas of teaching elementary school mathematics. *The elementary school journal*, 93 (4), 373-397.

Barab S. A., Kenneth H. E, Barnett M. & Keating T. (2000) Virtual Solar System Project: Building Understanding through Model Building. *Journal of Research in Science Teaching* **37**, 719-756.

Bell, Ph. (1995, April). How far does light go? : Individual and collaborative sense-making of scientific evidence. *AERA*, p. 1-36.

Bogdan, R., C. & Bilken,S., K. (1998). Qualitative research for education: An introduction to theory and methods. MA: Allyn & Bacon.

Colella, V., A., Klopfer, E., & Resnick, M. (2000). *Adventures in modeling: Exploring Complex, Dynamic systems with Starlogo.* NY: Teachers College Press.

Constantinou, C., P. (1996). The Cocoa microworld as an environment for modeling physical phenomena. *International Journal of Continuing Education and Life-Long Learning*, 8 (2), 65 - 83.

Cooper, S., Dan, W. & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15 (5), 107 – 116.

Creswell, W. J. (1988). Qualitative inquiry and research design: Choosing among five traditions. Thousands Oaks, CA: Sage Publications, Inc.

Cypher, A., & Smith, D. (1995). KidSim: End user programming of simulations. In *Proceedings of CHI' 95*, ACM Press, NY, p. 27-34.

Devi, R., Tiberghien, A., Baker, M., & Brna, P. (1996). Modelling students' construction of energy models in physics. *Instructional Science, 24,* 259–293.

diSessa, A. A., Abelson, H., & Ploger, D. (1991). An overview of Boxer. *Journal of Mathematical Behavior,* 10, 3-15.

diSessa, A., A. (1982). Understanding Aristotelian physics: A study of knowledge-based learning. *Cognitive Science*, 6, 37-75.

diSessa, A., A. (1988). Knowledge in pieces. In G. Forman & P. B. Pufall (Eds.), *Constructivism in the computer age*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Druin, A., Bederson, B., Boltman, A., Miura, A., Knotts-Callahn, D. & Plat, M. (1999). Children as our technology design partners. In A. Druin (Ed.). *The Design of Children's Technology*. San Francisco: Morgan Kaufmann Publishers, Inc.

Edwards, D. & Mercer, N. (1995). Common knowledge: The development of understanding in the classroom. NY: Routtedge.

Erduran, S. (1999) Philosophy of chemistry: an emerging field with implications for chemistry education. *Paper presented at the Annual Conference of the American Educational Research Association*, San Diego, 13–17 April.

Francoeur, E. (1997) The forgotten tool: the design and use of molecular models. *Social Studies of Science*, 27, 7–40.

Gallas, K. (1995). Talking their way into science: hearing children's questions and theories, responding with curricula. NY: Teachers College Press.

Gilbert, J. (1994). Models and modelling: routes to more authentic Science education. *International Journal of Science and Mathematics Education*, 2, 115-130.

Gilbert, J. (1995) The role of models and modelling in some narratives in science learning. Presented at the Annual Meeting of the American Educational Research Association, April 18-22. San Francisco, CA, USA.

Gilbert, J. K., Boulter, C. and Rutherford, M. (1998) Models in explanations, Part 1: Horses for Courses. *International Journal of Science Education*, 20, 83–97.

Glynn, S. M. and Duit, R. (1995) Learning science meaningfully: Constructing conceptual models. In S., M. Glynn & R. Duit (Eds.), *Learning science in the schools: Research reforming practice.* NJ: Lawrence Erlbaum Associates.

Glynn, S. M., Law, M., Gibson, N. M. and Hawkins, C. H. (1994) *Teaching Science with Analogies: A Resource for Teachers and Textbook Writers* (Instructional Resource No. 7) (Atlanta: National Reading Research Center, University of Georgia).

Gobert, J., D, & Buckley, B., C. (2000). Introduction to model-based teaching and learning in science education. *International Journal of Science Education*, 22 (9), 891-894.

Grosslight, L., Unger, Chr., Jay, E. and Smith, C., L. (1991) Understanding models and their use in science: Conceptions of middle and high school students and experts. *Journal of Research in Science Teaching*, 28 (9), 799-822.

Hammer, D. M., & Elby, A. (2003). Tapping epistemological resources for learning physics. *Journal of the Learning Sciences, 12*(1), 53-90.

Harrison, A., G. and Treagust, D., F. (1998) Modeling in science lessons: Are there better ways to learn with models? *School Science and Mathematics*, 98 (8), 420-429.

Hogan, K., & Thomas, D. (2001). Cognitive comparisons of students' systems modelling in ecology. *Journal of Science Education and Technology, 10*(4), 319–345.

Idling, M. E. (1997) How analogies foster learning from science texts. *Instructional Science*, 25, 233–253.

Ingham, A. M. and Gilbert, J. K. (1991) The use of analogue models by students of chemistry at higher education level. *International Journal of Science Education*, 13, 193-202.

Jonassen, D. H., Strobel, J., & Gottdenker, J. (2005). Modelling for meaningful learning. Learning Sciences and Technologies Group (Ed.), *Engaged learning with emerging technologies.* (pp. 7–28). Dordrecht, The Netherlands: Springer Verlag.

Justi, R. S. & Gilbert, J, K. (2002). Modelling, teachers' views on the nature of modelling, and implications for the education of modellers. *International Journal of Science Education* 24 (4), 369–387

Kahn, K. (1996). ToonTalk™. An animated programming environment for children. *The Journal of Visual Languages & Computing*, 7, 197-217.

Kelly G.J., Druker, S., & Chen, D. (1998). Students' reasoning about electricity: combining performance assessments with argumentation analysis. *International Journal of Science Education*, 20, 849-871.

Ko, A. J., Aung, H. H., Myers, B. A. (2005). Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective

Maintenance Tasks. In *the_Proceedings of the International Conference on Software Engineering* (pp.126-135). IEEE Computer Society.

Kurtz dos Santos A. C. & Ogborn J. (1994) Sixth form students' ability to engage in computational modelling. *Journal of Computer Assisted Learning* 10, 182–200.

Linn M. C. (2003) Technology and science education: starting points, research programs, and trends. *International Journal of Science Education* 25, 727-758.

Louca, L. (2004). Case studies of fifth-grade student modeling in science through programming: comparison of modeling practices and conversations. Unpublished doctoral dissertation, University of Maryland, College Park.

Louca, L. (2005). The syntax or the story behind it? A usability study of students' work with computer-based programming environments in elementary science. In *the Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 849-858). NY: ACM Press.

Louca, L. The use of programming environments by elementary students for scientific modeling: A comparison of Microworlds™ Logo and Stagecast Creator™. Paper submitted for peer review in Computers & Education.

Louca, L., Druin, A., Hammer, D., & Dreher, D. (2003). Students' collaborative use of computer-based programming tools in science: A Descriptive Study. In B. Wasson, St. Ludvigsen, & Ul. Hoppe (Eds.). *Designing for Change in Networked Learning Environments: Proceedings of the International Conference on Computer Support for Collaborative Learning 2003* (CSCL) (pp. 109-118) The Netherlands: Kluwer Academic Publishers.

Medawar, P. (1987). *Pluto's Republic*. Oxford: Oxford University Press.

Merriam, B. S. (1988). *Case studies research in education. A qualitative approach*. San Francisco, CA: Jossey-Bass, Inc., Publishers.

Miller, R., J. Ogborn, et al. (1993). Educational tools for computational modelling. *Computers & education* 21(3): 205-262.

National Research Council. (1990). *National Science Education Standards*. Washington, DC: National Academy.

Papaevripidou, M. Constsantinou, C. P. and Zacharia, Z. C. (in press). Modeling Complex Marine Ecosystems: Using Stagecast Creator™ to Foster Fifth Graders' Development of Modeling Skills. *Journal of Computer assisted Learning*.

Papert, S. (1980). *Mindstorms. Children, Computers & Powerful Ideas*. NY: Basic Books, Inc. Publishers.

Pea, R. (1984). *Intergrading human and computer intelligence*. Technical Report no. 32. Banks Street College of Education: New York, NY.

Penner, D. (2001). Cognition, Computers and synthetic science: Building knowledge and meaning through modeling. In Walter G. Secada (Ed.). *Review of Research in Education*. AERA: Washington, DC.

Penner, D., Lehrer, R., Schauble, L. (1998). From physical models to biomechanics: A design-based modeling approach. *The Journal of the Learning Sciences*, 7(3&4), 429-449.

Pittman, K. M. (1999) Student-generated analogies: another way of knowing? *Journal of Research in Science Teaching*, 36, 1–22.

Rader, C., Brand, C., & Lewis, Cl. (1997). Degrees of comprehension: children's understanding of visual programming environment, *Communications of the ACM*, 351-358.

Redish, E. F. & Wilson, J. M. (1993). Student programming in the introductory physics course: M.U.P.P.E.T. *American Journal of Physics*, 61 (3), 222-232.

Rouwette, E. A. J. A., Vennix, J. A. M., & Thijssen, C. M. (2000). Group model building: A decision room approach. *Simulation & Gaming, 31*(3), 359–379.

Schecker, H., P. (1993). The didactic potential of computer aided modeling for physics education. In D.L. Ferguson (Ed.). *Advanced Educational Technologies for Mathematics and Science*. NY: Springer-Verlag (NATO series)

Schoenfeld, A. H. (1989). Teaching mathematical thinking and problem solving. In L. B. Resnick & B. L. Klopfer (Eds.), *Towards the thinking curriculum: Current cognitive research* (pp. 83-103). Washington DC: ASCD.

Schwarz C. V. & White B. Y. (2005) Metamodeling Knowledge: Developing Students' Understanding of Scientific Modeling. *Cognition and Instruction* 23**,** 165-205.

Sheeham, R. (2004). The Icicle programming environment. In A. Druin, J. P. Hourcade & S. Kollet (Eds.), *Building a Community: The Proceedings of the Interaction, Design and Children Conference (IDC)* (pp. 261-267). New York, NY: The Association for Computer Machinery, Inc.

Sherin, Br. (1996). The Symbolic Basis of Physical Intuition. A Study of Two Symbol Systems in Physics Instruction. Unpublished dissertation Thesis.

Sherin, Br., diSessa, A. A., & Hammer, D. (1993). Dynaturtle revisited: Learning physics through collaborative design of a computer model. *Interactive Learning Environments*, 3 (2), 91-118.

Sinclair J. McH., & Coulthard, R., M. (1975). *Towards an analysis of discourse: The English used by teachers and pupils*. London: Oxford University Press.

Singer J., Krajcik J. & Marx R. (2000) The Design and Evaluation of Classroom Supports for Seamless Integration of a Dynamic Modelling Tool. In the proceedings of the *Fourth International Conference of the Learning Sciences* (eds B. Fishman & S. O'Connor-Divelbiss), pp. 62-69. Lawrence Erlbaum, Mahwah, NJ.

Singh, G., & Chignell, M., H. (1992). Components of the visual computer. *The Visual Computer*, 9, 115-142.

Sins, P. H., Savelsbergh E. R. & van Joolingen, W. R. (2005). The Difficult Process of Scientific Modelling: An analysis of novices' reasoning during computer-based modelling. *International Journal of Science Education*, 27, 1695–1721.

Smith, D., C. & Cypher, Al. (1999). Making programming easier for children. In A. Druin (Ed.). *The Design of Children's Technology*. San Francisco: Morgan Kaufmann Publishers, Inc.

Smith, D., C., Cypher, A., & Telser, L. (2000). Novice programming comes of Age. *Communications of the ACM*, 43 (3), 75-81.

Stake, R. E. (2000) Case Studies. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of qualitative research.* (435-454). Thousand Oaks, CA: Sage.

Strauss, A., & Corbin, J. (1998). Basics of qualitative research. Techniques and procedures for developing grounded theory. Thousand Oaks, CA: SAGE Publications.

Suthers, D. D. (1999). Effects of alternate representations of evidential relations on collaborative learning discourse. In C. M. Hoadley, & J. Roschelle (Eds.), *Proceedings of the Computer Support for Collaborative Learning (CSCL) 1999 Conference* (pp. 611–620). Palo Alto, CA: Stanford University.

Thompson, P., W. (1985, September). A Piagetian approach to transformation geometry via microworlds. *Mathematics Teacher*, 465-471.

Tomasi, J. (1988) Models and modelling in theoretical chemistry. *Journal of Molecular Structure*, 179, 273–292.

Treagust, D. F., Harrison, A. G., Venville, G. J. and Dagher, Z. (1996) Using an analogical teaching approach to engender conceptual change. International Journal of Science Education, 18, 213–229.

Underwood, G., Underwood, J., Pheasey, K., & Gilmore, D. (1996). Collaboration and discourse while programming the KidSim microworld simulation. *Computers & Education*, 26, 143-151.

White, B. Y. and Frederiksen, J. R. (1998). Inquiry, modeling and metacognition: Making science accessible to all students. *Cognition and Instruction*, 16 (11), 3-118.

Wilensky, Ur., & Resnick, M. (1999). Thinking in Levels: A Dynamic Systems Approach to Making Sense of the World. *Journal of Science Education and Technology*, 8 (1), 3-19.

Acknowledgements

We thank Dr. David Hammer for his insightful comments during the study's data collection and analysis and Dr. Wouter van Joolingen for his comments and suggestions during the manuscript preparation.

Footnotes

[1] Rules in Stagecast Creator are executed based on a rate that is defined by the software. Thus, unless defined, every software cycle one rule is executed.

Computer-Based Programming Environments as Modelling Tools

Figure captions

*Figure 1*. Program example from Microworlds Logo.

*Figure 2.* Program example from Stagecast Creator.

*Figure 3.* Samir's idea for their program.

*Figure 4*. Joe's idea for the program.

*Figure 5*. Microworlds student conversations while writing and debugging code.

*Figure 6*. Joe's and Samir's first program.

*Figure 7.* Microworld student activities while programming and debugging.

*Figure 8.* Conversation patterns during programming with Stagecast.

*Figure 9*. Stagecast Creator student activities while typing and debugging code.

*Figure 10*. Rules for a helium balloon.

*Figure 1*. Program example from Microworlds Logo.

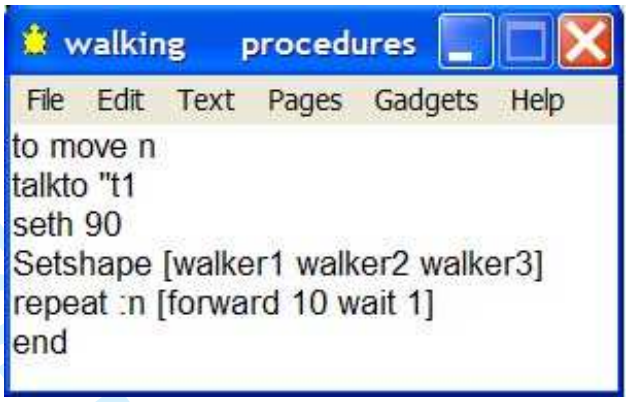


*Figure 2.* Program example from Stagecast Creator.

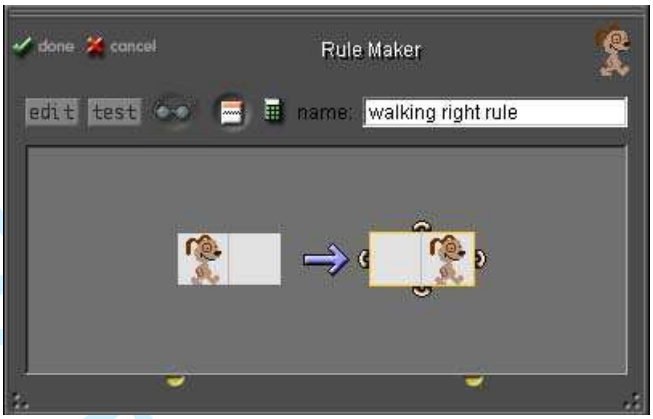
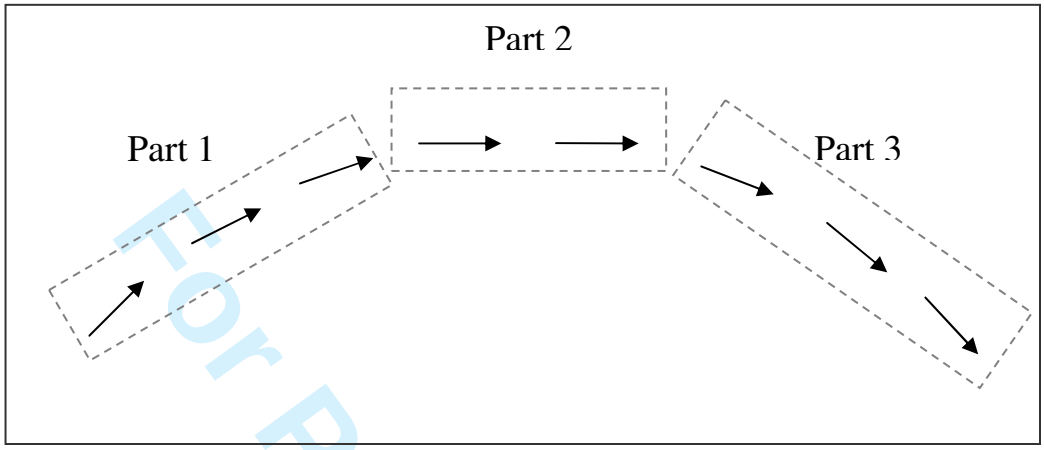


*Figure 3.* Samir's idea for their program.



Computer-Based Programming Environments as Modelling Tools

*Figure 4*. Joe's idea for the program.

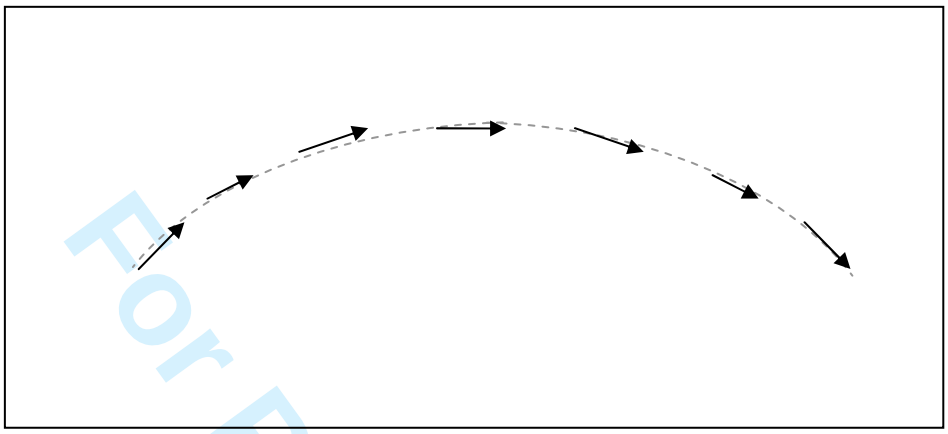


*Figure 5*. Microworlds student conversations while writing and debugging code.

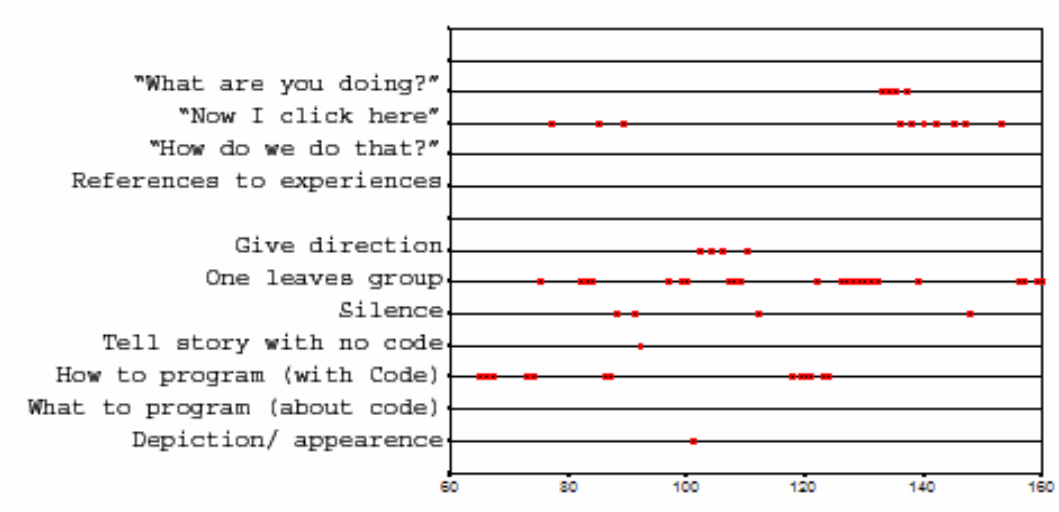


*Figure 6*. Joe's and Samir's first program.

```
to shoot
talkto "a
seth 90
local "angle
make "angle 45
repeat 30 [fd 5 lt 0.5 wait 0.1]
repeat 40 [fd 5 rt 1 wait 0.1]
end
```

*Figure 7.* Microworld student activities while programming and debugging.

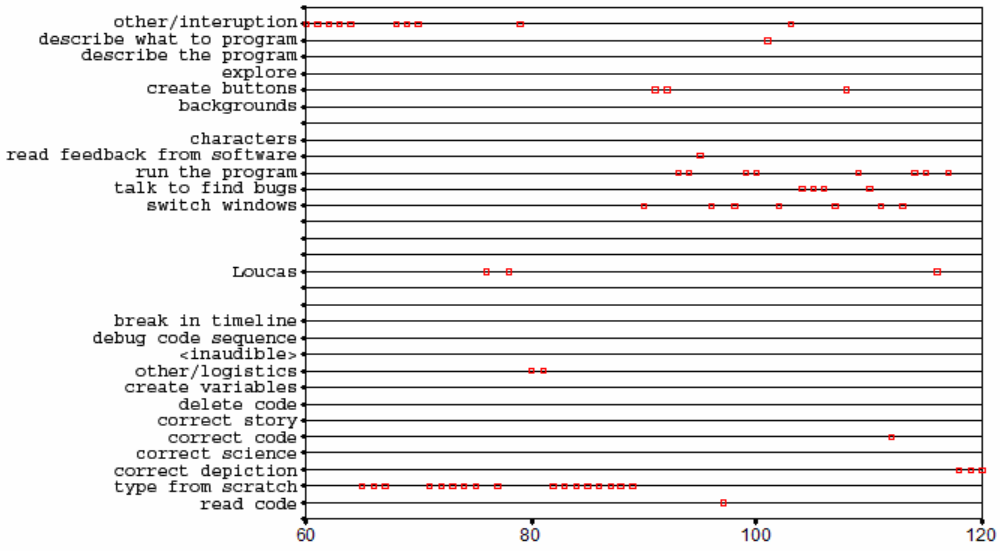


*Figure 8.* Conversation patterns during programming with Stagecast.

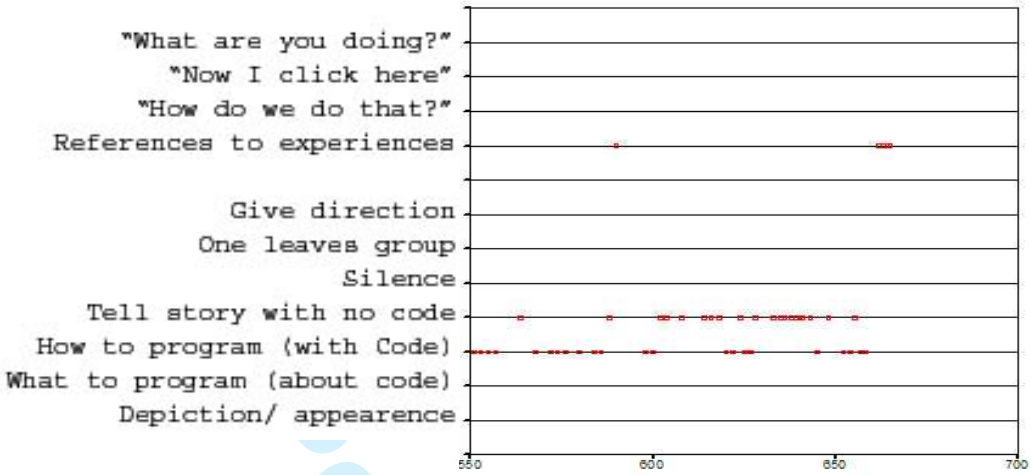


*Figure 9*. Stagecast Creator student activities while typing and debugging code.

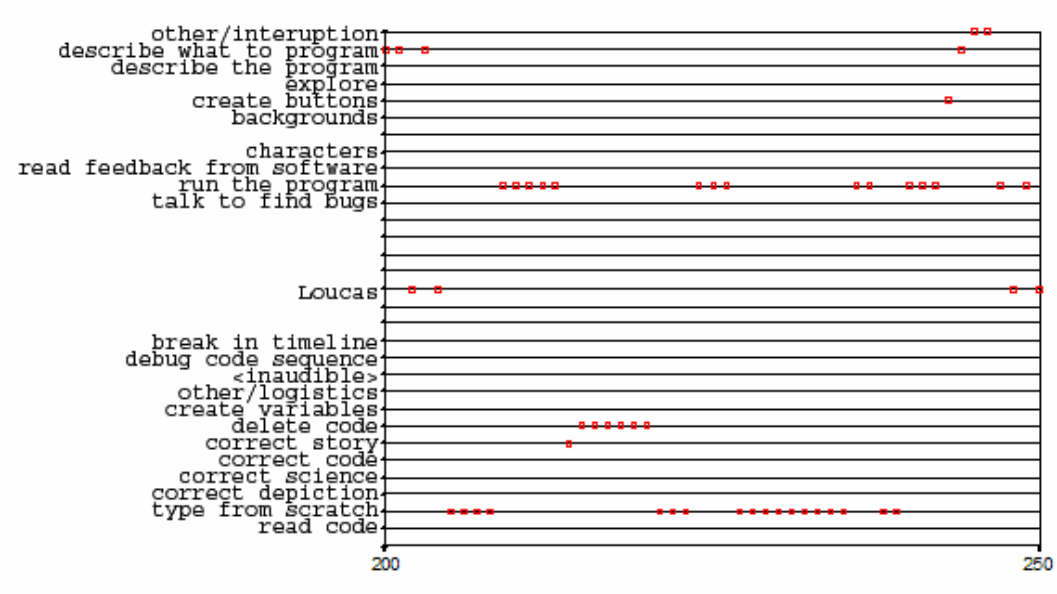
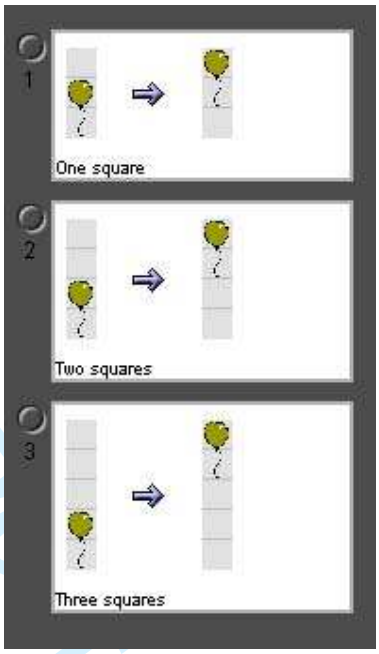


*Figure 10*. Rules for a helium balloon.



Computer-Based Programming Environments as Modelling Tools

*Table 1.* Comparison of features of CPEs based on the program language they use

| | Textual Programming Languages | Graphical Programming Languages |
|---|---|---|
| Example | Microworlds Logo | Stagecast Creator |
| Program process | Textual instructions | graphical before-after rules |
| Program strategies | write code | by demonstration |
| Programming | procedural | object oriented |
| Representation of objects | different representation in different modes | "analogical representation" |
| Representation of physical values | no differentiation | differentiation among variables & code |

*Table 2.* Summary of students' models during Study Part II

| Student groups | Models |
|---|---|
| Stagecast Creator Group 1 | Helium balloon game |
| Stagecast Creator Group 2 | Two athletes racing |
| Stagecast Creator Group 3 | A lake ecosystem |
| Stagecast Creator Group 4 | A ball falling down |
| Microworlds Logo Group 1 | The archer |
| Microworlds Logo Group 2 | A boy walking on a moving train |
| Microworlds Logo Group 3 | Collision of meteors |
| Microworlds Logo Group 4 | Jump on the Moon vs. on the Earth |

*Table 3*. Summary of findings

| Phases of student work | Characteristics of student work | |
|---|---|---|
| | Microworlds Logo | Stagecast Creator |
| Approaches to planning | • Students talked about their programs' structure<br>• Students had technical conversations<br>• Students saw their work as writing programs | • Students talked about the story they were about to program.<br>• Students broke down ideas in a number of sequential events<br>• Students saw their work as creating games |
| Approaches to writing and debugging code | • Students' goal was to type programs that would run<br>• Conversations were limited<br>• Initial programs consisted of a single routine & did not necessarily reflect their plans | • Student focus was on creating a simulation that would show their story<br>• Students talked about details of their scenario in an effort to translate them into programmable rules<br>• Debugging was deleting rules<br>• Initial programs were a number of rules that assign characters behaviors |
| Approaches to using code as the representation of the phenomenon itself | • The context of reading one's program was productive for modeling | • Tell the story of causal agents was productive for modeling |